

第4章

要求される条件を見定めてIPコアを設計する

── 仕様は明確に，回路のむだを省き，機能検証は怠らず

鈴木昌治

ここでは，前処理フィルタおよびヒストグラム生成などの計測用画像処理を実現するIPコアを例にとり，LSI内部のハードウェア設計について解説する．求められる機能を実現することはもちろんだが，製品としてのIPコアには「回路規模」，「性能」，「再利用性」などさまざまな条件も満たす必要がある．今回は，回路規模を抑えることを目標にして設計を行った．設計の過程やハードウェア実現のためのヒントなどに着目しながら読み進めてほしい．なお，本稿で紹介したIPコアの設計データは，本誌Webサイト（http://www.cqpub.co.jp/dwm/）からダウンロードできる．（編集部）

IPが「intellectual property（知的財産）」の略であることは，みなさんもご存じでしょう．ところで，LSI設計を行う際に「IPコア」ということばをよく耳にします．この場合のIPも知的財産の意味を持ちますが，ここではもう少し狭義に考え，IPコアを「ASIC（application specific integrated circuit）やFPGA（field programmable gate array）などにおいて，特定機能を提供するための回路設計情報」と定義することにしましょう．特定機能とは，例えばUART（universal asynchronous receiver transmitter）やPCI（peripheral component interconnect）などのインターフェース回路，SDRAMコントローラなどです．

Column　回路設計の「ココがすてき」

なんといっても回路が予定どおりに動作したときは最高です．コツコツと地味な作業を続けてきちんと結果を出す（セミが羽化したときって，こんな気分なんだろうか？）．そこに新しい挑戦が加わっていようものなら，それは至福のときです．

● 再利用性を考慮したIPコア設計が求められる

IPコアに求められる条件はさまざまです．基本機能を満たすことはもちろんですが，いろいろな状況の中で利用可能である（ユニバーサルである）ことが求められるケースも多いようです．それは，IPコアを利用する目的の一つとして，システム設計の効率化を図ることが挙げられるからです．しかし，この要求を満たそうとすると，設計はよりめんどうなものになります．

すべてが規格化されているような場合を除き，汎用のIPコアなどではバスやアドレスのビット幅，メモリの深さなどをある程度パラメータ化して制御できるようにしておかないと，むだの多い大規模な回路となってしまいます．

また，ユニバーサルに使おうと考えて設計されたものでも，作り込まれた機能がつねに必要とは限りません．例えば，SDRAMコントローラなどはその典型です．すべてのコマンドを駆使して複雑なアクセスを行うケースはごくまれで，特定のアプリケーションに限定すればアクセスのスタイルも限定される，という状況がほとんどではないかと思います．したがって，使わない機能は回路化されないようなくふうも必要になってきます．

1　具体例を見ながらIPコア設計を学ぶ

本稿ではIPコア設計の具体例として，画像処理機能を

KeyWord　IPコア，Stratix（EP1S10），空間フィルタ，レベル変換，ヒストグラム生成，ラスタ・スキャン，ライン・メモリ，データ・フロー制御，ステート・マシン，シミュレーション・モデル



（a） 処理前の画像



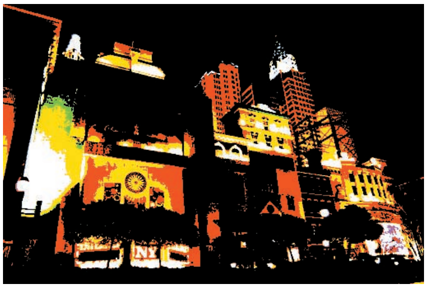
（b） 平滑化（8近傍移動平均フィルタ）



（c） エッジ検出（8近傍ラプラシアン・フィルタ）



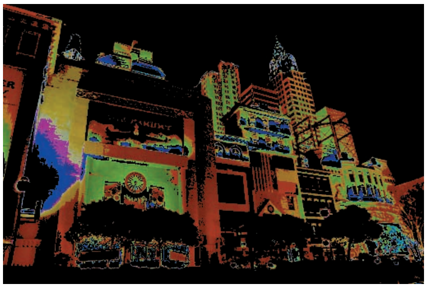
（d） 先鋭化（8近傍先鋭化フィルタ）



（e） 階調間引き（2値化）



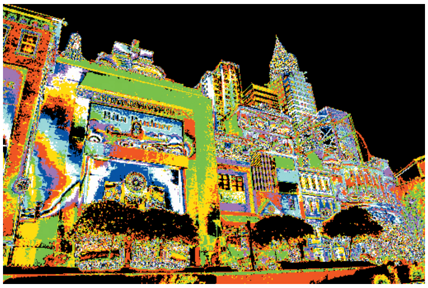
（f） 階調間引き（4階調）



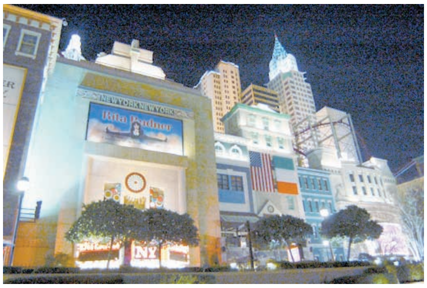
（g） 特定階調抽出



（h） 階調反転



（i） 等高線表示



（j） ヒストグラム平たん化

写真1　今回作った画像処理IPコアでできること

今回は，計測用画像処理のための前処理フィルタやレベル変換，およびヒストグラム生成を実現するIPコアを作った．計測用画像処理は特徴を効率良く取り出すために強烈な処理を施すことが多いので，見た目に効果がわかりやすい（RGBを個別に処理しているので，若干わかりにくい効果もあると思うが…）．

実現してみたいと思います．ただし，ここで紹介するのは先に述べたようなユニバーサルなものではありません．今回は，CQ出版社のStratix評価キットを利用しますが，本キットに搭載されているFPGA（米国Altera社のFPGA「EP1S10」）の規模をにらんだうえでおもしろい結果が得られるように，本稿に合わせて設計された特定用途のIPコアと考えてください．

● 今回の目標は「回路規模を小さく」

広義に画像処理と言うと，観賞用画像の生成を目的としたものと，画像から特徴を抽出することを目的としたものの二つに大別できるのではないかと思います．前者は，画像のフォーマット変換に伴う拡大/縮小や，画像効果のための変形などが良い例です．後者は，画像から物体の数を調べたり，動きを検出したり，形状を認識したりと，計測用途が主体となります．

ここで実現するのは，後者のための前処理フィルタ，レベル変換（階調変換）およびヒストグラム生成であり，画像計測の入門的な処理になります（**写真1**）．このような機能を選んだ理由は，観賞用画像処理が情報を損なわずに見た目のきれいさを重視するのに対し，計測用画像処理は特徴

を効率良く取り出すために強烈な処理を施すことが多いので，見た目に効果がわかりやすいからです．また，計測用画像処理は処理結果に対してホスト側でなんらかの判断を行い，それに基づいて別の処理を再度実行するなど，システムが有機的に機能すると考えたためです．

とはいっても，あくまでも前処理的な内容で実用性には乏しいものですし，この程度の処理であれば専用ハードウェアで実現する必要もないと思います．ここで紹介するものは，あくまでもハードウェア実現の入門のサンプルと考えていただき，実現したハードウェアそのものよりも実現に至る過程や実現のためのヒントなどに着目していただきたいと思います．

ハードウェアはソフトウェアと違い，処理系の並列度を上げることで処理能力を向上させることができます．逆に，並列度を上げずに時間軸方向でシーケンシャルな処理を行えば，回路規模を抑制できます．何の目的もなく回路を設計してもおもしろみが出てこないので，ここでは並列度を上げずにできる限り規模の小さい回路を作成することにしましょう．ディジタル・フィルタなどは数値演算回路の集合体なので，並列度を上げなければ規模を大幅に抑制できます．

● 設計の大前提は「最初に仕様をきっちり決める」こと

システムを構築するにあたり，仕様を決める作業は非常に重要な要素になります．

かつて，Verilog HDLやVHDLのような回路記述言語が台頭し始めたころ，「これでトップダウンに設計できる！」などと喧伝されたものですが，トップダウン設計の最上流である仕様決めをこれらが解決することはありませんでした．現在では仕様を記述する言語やツールもありますが，これらはあくまでも道具でしかありません．「人が真剣に考えることで仕様ができ上がるのだ」と，まずは肝に銘じておきましょう．明確な仕様を決めずにRTL（register transfer level）で回路を記述し始めてしまう先輩エンジニアもいるかもしれませんが，それは“論外の手順”と考えてください．

- **仕様はできるだけ詳しく決めること**
- **決めた内容に対する根拠（なぜそう決めたか）をドキュメントに残しておくこと**

がその後の作業で重要なポイントとなります．

● フィルタ処理，レベル変換，ヒストグラムを実現する

さて，前置きはこのくらいにして本題に入りましょう．

本特集用の画像処理IPコアを設計するにあたって初期の段階で決めた仕様を**図1**に示します（仕様決めの過程については，pp.84-90のAppendix「実録『画像処理システムができるまで』」を参照）．また，コマンド定義とI/O定義，および制御パラメータはそれぞれ**図2**，**表1**，**表2**（pp.68-69）に従うものとします．

- IPコアの動作クロックは66MHz（＝SDRAMクロック）とする
- IPコアから見た画像データの入力元（ソース領域），出力先（デスティネーション領域）はいずれもSDRAM上とする
- 実行コマンドをパラメータとして設定した後，スタート・パルスを与えることで処理を開始する
- 処理を開始できることを知らせるready信号をIPコアからトップ・レベルに返し，処理実行中は再スタートさせない．ただし，再スタート信号により，実行中のコマンドを中止させることができる
- 処理実行中はIPコアがSDRAMの制御権を獲得し，ポートを占有する
- 制御パラメータはPCIバス・ブリッジ（PCIターゲット）から設定され，プロトコルはPCIバス・ブリッジIPコアに従う（詳細はSDRAMコントローラ・インターフェース・モジュールにて説明）
- 画像データのフォーマットはRGB24ビットとし，R［23：16］，G［15：8］，B［7：0］とする
- 画像サイズは1画面で最大1,024×512画素とし，SDRAM上に4面確保する（4面の指定は図2を参照）．ただし，面の指定を偶数とすることで，最大1,024×1,024サイズ2面として使用することもできる

図1　設計初期段階で決まった仕様
設計を始めるにはまず「仕様」をきちんと決めておく．

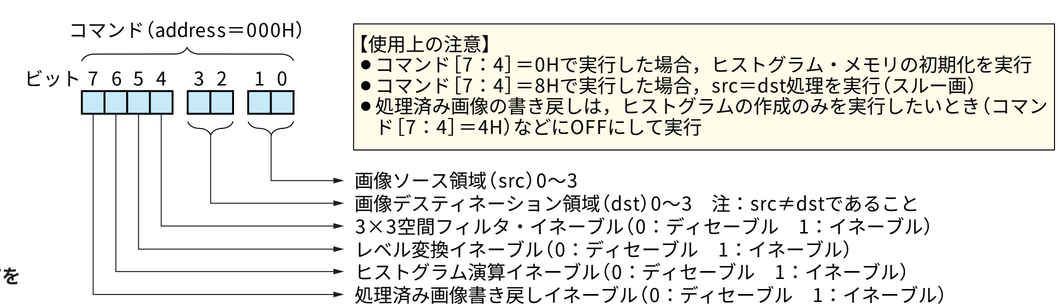


図2
コマンド定義
コマンド8ビットの機能割り当てを図のように取り決めた．

今回の画像処理IPコアで何を実現できるかについて，ざっくりとお話ししておきましょう．

簡単なアプリケーション例を，**表3**に示します．画像に変化を与える要素としては，「3×3フィルタ」と「レベル変換」の二つがあります．さらに，画像が持つ情報を抽出する要素として「ヒストグラム」があります．

3×3フィルタは，3×3の格子の中心画素とその周辺画素に係数を乗算し，和をとることで，空間フィルタを構成します．画像計測の前処理フィルタとして重要な役割を果たします．係数の設定しだいでいろいろな機能を持たせる

表1 I/O定義

分 類	名 称	ビット幅	I/O	説 明
システム制御信号	clk	―	入力	クロック66MHz
	rst_x	―	入力	非同期パワーONリセット（アクティブ“L”）
	hot_restart	―	入力	同期ホット・リセット（アクティブ“H”，1パルス）
	exec_start	―	入力	スタート・コマンド実行（アクティブ“H”，1パルス）
	exec_ready	―	出力	コマンド実行レディ（not busy）（アクティブ“H”）
SDRAM制御	sdram_a	[20：00]	出力	2Mワード，ワード・アドレッシング
	sdram_d_rd	[31：00]	入力	SDRAM読み出しデータ
	sdram_d_wr	[31：00]	出力	SDRAM書き込みデータ
	sdram_rd_x	―	出力	SDRAMアクセス・モード（リード）（アクティブ“L”）
	sdram_we_x	[03：00]	出力	SDRAMアクセス・モード（ライト）（アクティブ“L”）
	sdram_req_x	―	出力	SDRAMアクセス要求（アクティブ“L”）
	sdram_ack_x	―	入力	SDRAMアクセス成功（アクティブ“L”）
PCIパラメータ制御	param_a	[10：00]	入力	2Kワード，ワード・アドレッシング
	param_d_rd	[31：00]	出力	読み出しデータ
	param_d_wr	[31：00]	入力	書き込みデータ
	param_wr_x	―	入力	アクセス・モード（リード=“H”，ライト=“L”）

表3 作成した画像処理IPコアのアプリケーション例

3×3空間フィルタ	LPF（ノイズの除去など）
	HPF（エッジ検出など）
	先鋭化
	そのほか（係数しだい）
レベル変換	階調間引き（2値化含む）
	階調反転
	特定階調抽出
	等高線表示
	ガンマ補正
	コントラスト補正
	そのほか（テーブルしだい）
ヒストグラム	ヒストグラム作成

0	1/5	0
1/5	1/5	1/5
0	1/5	0

（a）4近傍移動平均フィルタ（LPF）

0	1	0
1	−4	1
0	1	0

（b）4近傍ラプラシアン・フィルタ（HPF）

0	−1	0
−1	5	−1
0	−1	0

（c）4近傍先鋭化フィルタ

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

（d）8近傍移動平均フィルタ（LPF）

1	1	1
1	−8	1
1	1	1

（e）8近傍ラプラシアン・フィルタ（HPF）

−1	−1	−1
−1	9	−1
−1	−1	−1

（f）8近傍先鋭化フィルタ

図3 代表的な3×3フィルタ係数

目的に応じた係数を設定することで，フィルタの機能が変わる．

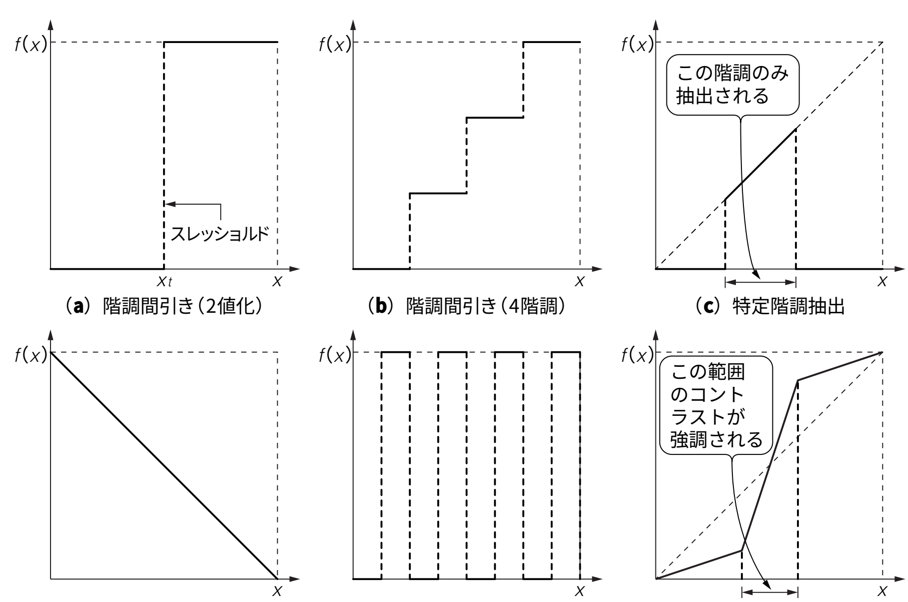


図4 代表的なレベル変換テーブル

目的に応じたテーブルを設定することで，変換の機能が変わる．

表2　レジスタ設定

W/Rの欄は，そのアドレスがwrite onlyまたはread onlyであることを示す．データ・バスは32ビット幅だが，使用する24ビットのみを示している．

モジュール	設定値名	ビット幅	説　明	W/R	注意事項
トップ・レベル	command	8	実行コマンド	W	[7：4]：実行コード，[3：2]：dst，[1：0]：src
	h_size	11	処理領域Hサイズ	W	無符号整数
	v_size	11	処理領域Vサイズ	W	無符号整数
3×3 空間フィルタ	coef_flt_00	16	空間フィルタ係数00	W	[15]：符号，[14：10]：整数部，[9：0]：小数部
	coef_flt_01	16	空間フィルタ係数01		
	coef_flt_02	16	空間フィルタ係数02		
	coef_flt_10	16	空間フィルタ係数10		
	coef_flt_11	16	空間フィルタ係数11		
	coef_flt_12	16	空間フィルタ係数12		
	coef_flt_20	16	空間フィルタ係数20		
	coef_flt_21	16	空間フィルタ係数21		
	coef_flt_22	16	空間フィルタ係数22		
ヒストグラム 生成	hist_r	20	ヒストグラムR度数　階調0 ≀ ヒストグラムR度数　階調255	R	無符号整数
	hist_g	20	ヒストグラムG度数　階調0 ≀ ヒストグラムG度数　階調255	R	無符号整数
	hist_b	20	ヒストグラムB度数　階調0 ≀ ヒストグラムB度数　階調255	R	無符号整数
変換テーブル	table	24	変換テーブル値　階調0 ≀ 変換テーブル値　階調255	W	無符号整数，R[23：16]，G[15：8]，B[7：0]

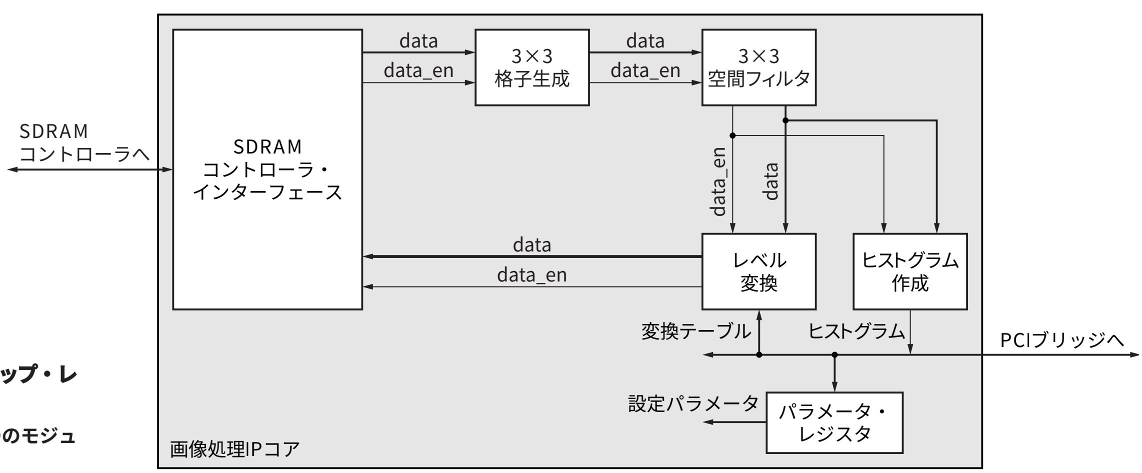


図5
画像処理IPコアのトップ・レベル・ブロック図
機能実現のための六つのモジュールから構成される．

ことができます．**図3**に，機能実現のための係数例を示します．いずれも著名なフィルタです．フィルタをかけた結果については，**写真1(b)～(d)**で確認してください．フィルタについての詳細は専門の参考書にゆずることとし，およそこんな機能だということを理解してください．

レベル変換は非常に便利な機能です．今回は，演算ではなくテーブルを参照する方法で実現するため，正直なところ何でもありです．テーブルは1系統当たり8ビット×256のメモリで十分なので，今回使用するFPGAであれば問題なく実現できます．

$$x' = f(x) \quad \cdots\cdots(1)$$

オフセット・アドレスA［12：2］	データ［23：0］																							
	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
000																	command							
001														h_size										
002														v_size										
010									coef_flt_00															
011									coef_flt_01															
012									coef_flt_02															
013									coef_flt_10															
014									coef_flt_11															
015									coef_flt_12															
016									coef_flt_20															
017									coef_flt_21															
018									coef_flt_22															
100					hist_r［0］																			
〜																								
1FF					hist_r［255］																			
200					hist_g［0］																			
〜																								
2FF					hist_g［255］																			
300					hist_b［0］																			
〜																								
3FF					hist_b［255］																			
400	table_r［0］								table_g［0］								table_b［0］							
〜																								
4FF	table_r［255］								table_g［255］								table_b［255］							

という関係さえ定義できれば，それをテーブル・メモリにホストから書き込むだけなので，応用範囲はかなり広いものになります．**写真1（e）〜（i）**に示したものもほんの1例にすぎません．各効果を得るためのテーブルに設定する関数を**図4**（p.67）に示します．

ヒストグラムは，今回実現するシステムにおいて唯一の計測機能になります．ヒストグラムは統計手法の基本的なもので，画像処理においては階調（濃度）の分布を調べるために用いられます．例えば，全体に暗くて何が写っているのかわかりにくい画像があったとします．この画像の濃度のヒストグラムをとることで，濃度がどのあたりに偏っているかを判別します．偏っている箇所の濃度をレベル変換を用いて拡大すれば，見やすい画像になります．これをヒストグラムの平たん化と言います（**写真1（j）**）．画像処理の基本手法の一つですが，コントラスト補正の一種と考えてもよいでしょう（詳細はp.83のコラム「ヒストグラムの平たん化の方法」を参照）．

2 IPコア全体の構造と制御方法を決める

ここまで説明したような機能を実現するシステムのトップ・レベルのブロック図を**図5**に示します．この図に沿って，画像処理IPコアの大まかな動作を説明します．

● パソコンとIPコアの間の処理の流れ

今回，処理を行う画像データは，ホスト（パソコン）によってPCIバス経由でSDRAM上に書き込まれます．パラメータとして処理内容を示すコマンド・コードと画面サイズが設定され，必要に応じてそのほかのパラメータも設定したうえで，スタート・パルスを与えます．スタート・パルスによって各モジュールはフレームの初期化が行われ，必要な処理を開始します（**図6**）．

「SDRAMコントローラ・インターフェース→3×3格子生成→3×3空間フィルタ→レベル変換→SDRAMコントローラ・インターフェース」のパスは，画像データの処理パスになります．SDRAMコントローラは，画面サイズに従って必要な画素数分だけ順次SDRAMからデータを読み出し，後段へ送ります．データを受けた後段は，能動的にデータ処理を開始します．「能動的に」という意味は，データ処理に必要なデータ数がモジュールによって異なるので，各モジュールでそこを判断して処理を行うことを指します．モジュールごとに入力データを処理し，データ・イネーブル（data_en）とともに後段に渡していきます．最終的にSDRAMコントローラ・インターフェースに至ったデータはSDRAMに書き込まれ，画面サイズで設定された画素数を処理した後，コマンドの実行を終了します．SDRAMに書き込まれたデータは，ホストによってPCIバス経由で読み出され，再利用されます．

「3×3空間フィルタ→ヒストグラム作成」のパスは，フィルタ通過後のデータに対してヒストグラムを作成するパスになります．1フレームに対してRGB別にヒストグラムを作成し，結果の読み出しはPCIバス経由でホストが読み出します．

● データ・フロー制御はモジュールの処理速度を考えて

処理の並列度の違いや処理内容の違いにより，1画素当たりの処理サイクルがモジュールによって異なります．したがって，データを上流からたれ流しにするだけでは，処

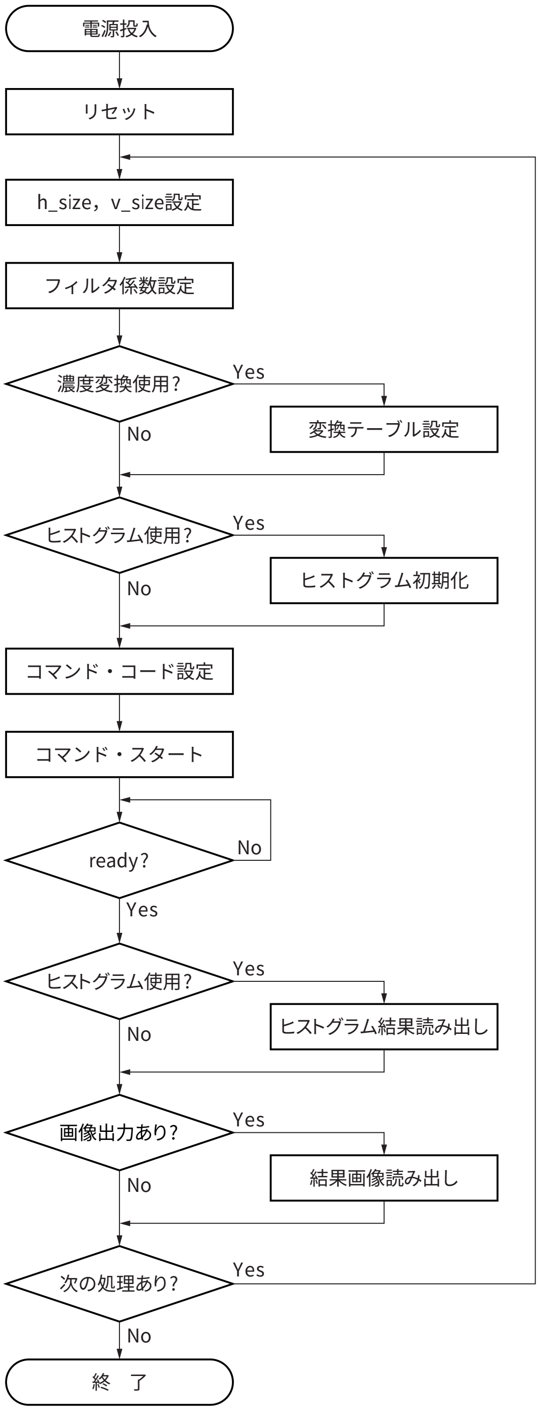


図6　画像処理IPコアの使用手順
本画像処理IPコアの使用手順をフローチャートで示した．

理が追いつかないモジュールが出てきます．もっとも処理の遅いモジュールに合わせてデータを流すレートを決めてやればよいのですが，未知のIPコアが上流にあったり，仕様変更によって処理サイクルが変わったりと，設計初期の段階ではどれが遅いモジュールになるかわからない場合が少なくありません．また，設定によっては処理速度に変化がある場合も想定しなければなりません．そのようなときにはどのように処理を行えばよいのでしょうか．

解決法の一例を，水の流れにたとえて説明してみましょう（**図7**）．今回は処理の対象になるデータは，ホストによってSDRAM上に書き込まれているという前提があります．**図7**の例で言えば，雲（ホスト）から雨が降ると（書き込み動作が発生すると），池（SDRAM）に水（データ）が貯えられている，ということになります．

水門（SDRAMコントローラ・インターフェース）の貯水タンク（リード側FIFOメモリ）には，暇さえあれば水をつねに貯えるようにします．蛇口（3×3格子生成）を開いたら，すぐに水を使える状態にしておくためです．ただし，タンクがあふれない程度に水門を加減する必要があります．つまり貯水タンクとしては，自分のタンクの貯まり量によって水門の開閉を制御するだけでよいわけです．

この貯水タンクに水がある場合は蛇口を開いて水を取り込み，次の水道管A（3×3フィルタ）のために水を用意します．しかし，水道管Aは水の消費が遅い（データの処理速度が低い）ため，一度水を受け入れてしまうと次に水が必要になるまでに少し時間がかかります．これを無視して水を送り出してしまうと，水が水道管Aであふれることになります．よって，ここでは貯水タンクに水があり，水道管Aが次の水を欲しがっている場合に蛇口を開くことにします．これで適量を送り出すことになります．

水道管A以降も別の蛇口を設けることで水の流れを制御できますが，**図7**では水道管Bに直接つながっています．これは，今回は3×3フィルタがもっとも処理速度が遅いとあらかじめわかっているからです．先ほど「設計初期の段階ではどれが遅いモジュールになるかわからない」と言っておきながら何なのですが，この条件を適用することで3×3フィルタ以降をたれ流し制御とし，制御の簡素化を図っています．

水道管を通って最後にデータの行き着く貯水タンク（SDRAMコントローラ・インターフェースのライト側FIFOメモリ）ですが，ここから水を池に戻すことになります．ここで，図では池から水を取り出す経路と池に戻す経路は1本の川になっています（SDRAMがシングルポートであるため）．そのため，いずれか一方向にしか水は流れないことになります．水を池から取り出せなければ，水道管は水が来るのを待つだけです．しかし，水を池に戻せな

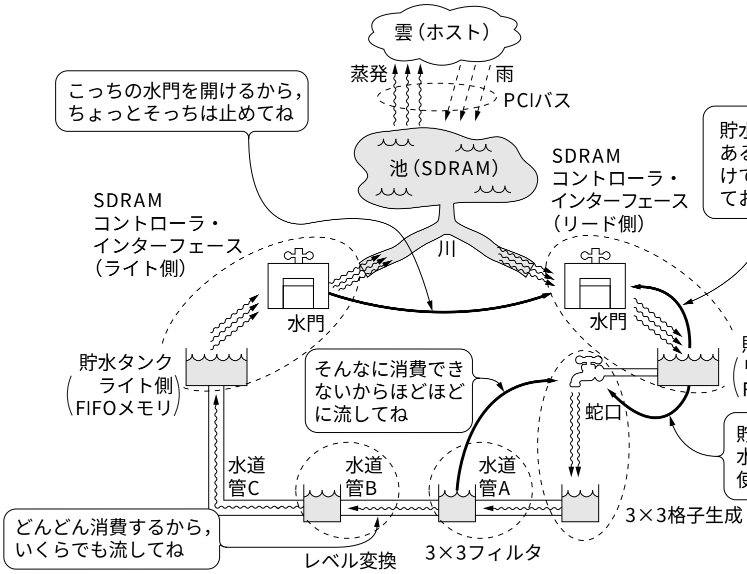


図7　画像処理IPコアの制御を水の流れにたとえてみると…
太い矢印で示した部分が必要とされる制御を示す．

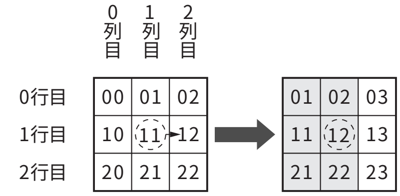


図8　3×3格子におけるデータの再利用
各画素を図のように定義して中心画素を右に1画素分移動する．灰色に塗られている6画素はさっき使ったデータなので，再度SDRAMから読み出すのではなく再利用したい．

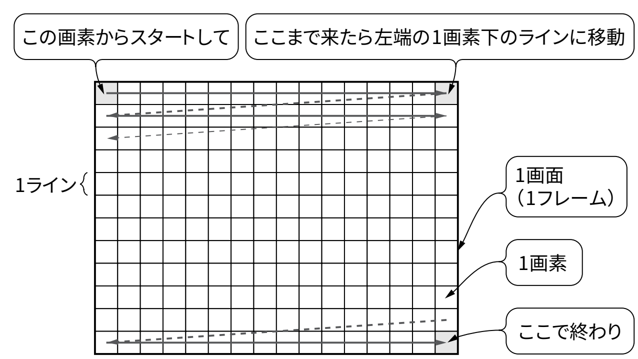


図9　ラスタ・スキャン
テレビなどでおなじみの走査方法．2次元情報をこのように走査することで，1次元情報に変換する．

い場合，貯水タンク（ライト側）には上流から供給される水を止める機能がないので，あふれてしまうかもしれません．したがって，貯水タンク（ライト側）に水が貯まったときには，水を池に戻すことが優先されます．こうして，池に戻った水は蒸発して雲（ホスト）に戻っていきます．

以上が，全体のおおよその制御となります．データ（水）がどこの条件で流されたり止められたりするかをつかみ取ってください．

3 各モジュールの処理内容

ここまでで全体の構造と制御が決まったので，今度は下位の機能モジュールを決定する過程に入ります．

● 3×3フィルタ用格子生成 ―― データの再利用を考慮

3×3フィルタ演算を行うには，中心画素と周辺画素の9画素のデータを用意する必要があります．その画素データを準備して3×3格子としてフィルタに渡すのが，このモジュールの機能になります．同時に，SDRAMインターフェースのリード側FIFOメモリからデータを取り出す蛇口となることは先に述べたとおりです．

1）横方向の画素データの扱い

9画素が必要なのはわかりましたが，1画素の出力を得るために，毎回周辺画素も含めて9画素を準備するのは効率が悪い，ということは直感的に理解できるでしょう．とくにSDRAMはSRAMの場合と違い，アクセスはバースト・サイズとその方向によって制約を受け，3×3格子を取り出すには不向きです．中心画素を1画素右にずらした場合，9画素のうち6画素はオーバラップしているわけなので，読み出したデータをできる限り再利用することを考える必要があります（**図8**）．これを解決するのが，ラスタ・スキャンとライン・メモリを用いた手法です．

画素データはフレーム（画面）の左上端の画素から水平右方向へ順に取り出し，右端に到達したら1画素下の左端に戻って，同じ操作を繰り返します．これをラスタ・スキャンといいます（**図9**）．テレビの場合もそうであるように，このような画素の走査方法はごく一般的で，3×3格子生成用の特別な手法というわけではありません．横方向のデ

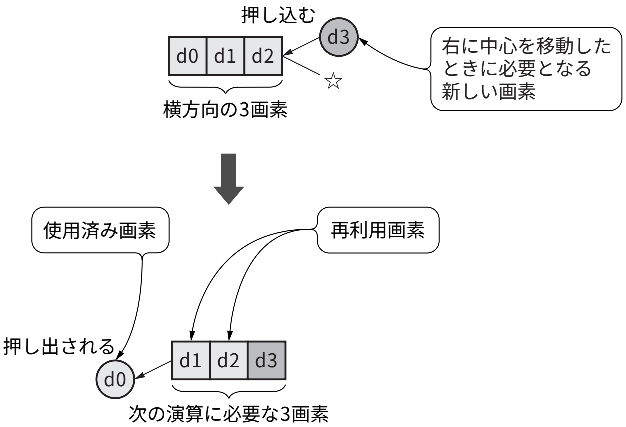


図10　シフト・レジスタによる水平方向のデータ再利用
シフト・レジスタにより，再利用できるデータを保存しておく．

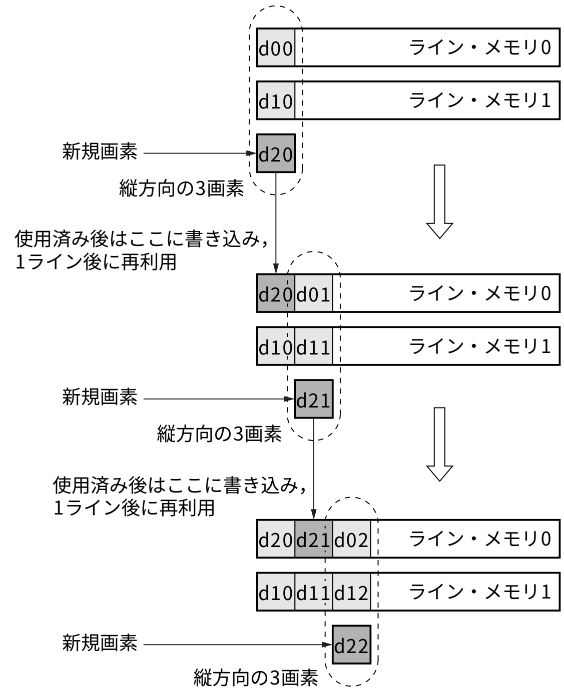


図11　ライン・メモリによる垂直方向のデータ再利用
水平方向の走査を行った場合，垂直方向のデータを再利用するためには，ライン・メモリが必要になる．

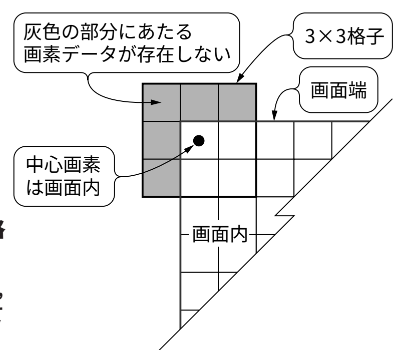


図12
画面端における3×3格子データの欠如
3×3格子を構成するには，画面端でデータが欠如してしまう．

ータの再利用は，このように横方向にスキャンしていくことを前提に行われます．横方向の3画素データを3段のシフト・レジスタに格納しておき，次の画素に移る段階で新しい1画素データをシフト・レジスタに押し込んでやれば，使用済みとなった古いデータが押し出されます．再利用可能な2画素分のデータはシフト・レジスタ上に残り，新しい3画素データが得られるというわけです(**図10**)．

2）縦方向の画素データの扱い

以上で横方向のデータの再利用はできましたが，3×3格子生成には，縦方向にも3画素のデータが必要になります．

任意のライン上の画素データは，そのラインを中心画素とする以外に，もう2回，周辺画素としての役割があります．つまり，合計3ラインのフィルタ演算に登場することになります．これは，横方向のデータの再利用と同じ考えかたです．異なるのは，横方向にスキャンしている関係上，次に再利用されるのは1ライン後になるということです．

したがって，新規データとしてSDRAMから読み出したときに1回使用されるとすれば，もう2回使用するには2ライン分のデータを保存しておく必要があります．このデータを保存するメモリをライン・メモリと呼び，これを用いて**図11**のようにデータを取り回します．ライン・メモリ0に新規に読み出されたデータd20，d21，d22が，次のラインで再利用されるために保存されていくようすに注目してください．

以上の縦と横のデータの再利用を併用することで，つねに新規に読み出すデータを1画素としながら，3×3格子が生成されます．

3）画面端の処理

さて，ディジタル・フィルタを使う際にかならず直面する問題が，画面の端における処理です．**図12**に示すのは，中心画素が左上端にある場合です．3×3格子における右下側の4画素は画素データが存在するのですが，左上側(図の灰色の部分)の5画素は画素データが存在しません．ここにはどのようなデータを入れておくべきかが問題となるわけです．

多くの場合，あたり障りのない適当なデータとして，黒データが入れられているようですが，この方法には次のような欠点があります．

- 端に行くほど採用された色(黒)がにじんでくる

● フィルタ係数の設定によってリンギングが発生する

タップ数の大きいフィルタの場合，影響を受ける範囲が広くなるので，深刻な問題となります．画像計測の場合，特徴を左右する重要な要素を画面の端にとらえることはあまりないことですし，3×3格子程度であればさほど問題とされません．しかし，フィルタを繰り返しかけていくと，その影響は内側に広がってくることになります．

今回の事例ではこの対策として，画面の外側に鏡像をイメージする方法を採用しました．**図13**のように，画面の外側に鏡像があるかのように扱って，画素が存在しない部分に画面内側の有効な画素をコピーして使用するのです．これによって，画素のない部分がより適切な画素で満たされるので，自然な画面端が得られます．また，ソース画面の中にありえなかった階調が発生するのを防ぐこともできます．

以上の点に基づき，1フレームにおける処理シーケンスを**図14**に示します．画素データの再利用という観点からすれば2ライン・シーケンスになるのですが，画面端の処理を加えたことで1フレーム・シーケンスとなります．1フレーム・シーケンスの実現にVカウンタ(vc；縦用カウンタ)，1ライン・シーケンスにHカウンタ(hc；横用カウンタ)，1画素シーケンスにc02というカウンタを用意して，3階層のシーケンサとなっています．vcとhcはスタート・パルスで‘0’にリセットされた後，hcは1画素に1回，vcは1ラインに1回のインクリメント(＋1)を繰り返し，v_size，h_sizeのパラメータで設定された画面のサイズに到達するまで3×3格子を生成し続けます．こうして1フレーム分の処理を終えた後，vcはカウンタの最大値にセットされます．画面サイズを前回使用したサイズより大きな値に変更したとき，vc＜v_sizeとなって，スタート・パルスを待たずに動作を開始してしまうのを避けるためです．

ここまででの処理を実現する構造のブロック図を**図15**(p.75)に示します．3×3格子生成は図の左側のブロックになります(詳細はRTL記述を参照)．

● 3×3フィルタ――回路規模を抑えるくふうを施す

3×3フィルタは，式(2)で示される演算を行うだけです．

$$x' = \sum_{i=0}^{2}\sum_{j=0}^{2} x_{ij} \times cf_{ij} \quad \cdots\cdots (2)$$

xは画素の階調，cfはフィルタ係数を示し，添字は3×3格子の位置を示しています(**図8**を参照)．誌面のつごう上，ディジタル・フィルタの詳細の説明は割愛します．式(2)が表しているのは，シーケンシャルに解釈すれば，3ステップの`for`文をネスティングして9回の乗算と累積を行っているということです．

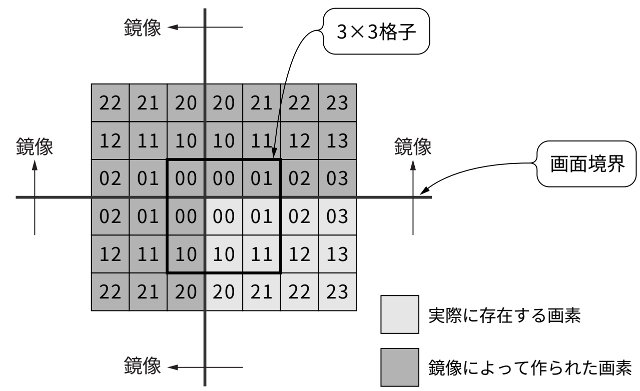


図13　画面端における欠如データの補修
鏡像を画面の外側にイメージすることで，欠如したデータを補う．

今回実現する構造は，ハードウェアの並列化を避け，回路規模をケチる(抑える)ことにしているので，シーケンシャルな表現を採用し，一つの乗算器で9回の乗算を行い，一つの加算器で累積演算を行うこととしました．さらに，そのシーケンスを3回繰り返すことで，RGB3色の処理を行うことにしたので，合計27クロックのシーケンスになります．1色当たりの9クロック・シーケンスにはc04(RTLではc02a)を，3色のための3回の繰り返しにはc02(RTLではc02b)を割り当てて，2階層のシーケンサを構成しています．

ところで，フィルタをかけない場合も27クロックかけてしまうのでは，時間のむだであまりにくふうがないと感じられるかもしれません．その場合，中心画素のフィルタ係数を1として，中心画素を1倍する操作のみを行うことでフィルタをかけない状態を作っています．このため，中心画素とフィルタ係数の乗算を9クロック・シーケンスの最終ステートに割り当て，フィルタをかけない場合には最終ステートにじかに到達して，1クロック・シーケンスとなるようにしています．

ただし，RGB3色のための3クロック・シーケンスはそのままなので，フィルタをかけない場合でも最低3クロックかかることになります．これにより，フィルタをかけない場合でも，3×3フィルタはもっとも処理の遅いモジュールになるようにしています．これは，トップ・レベルの

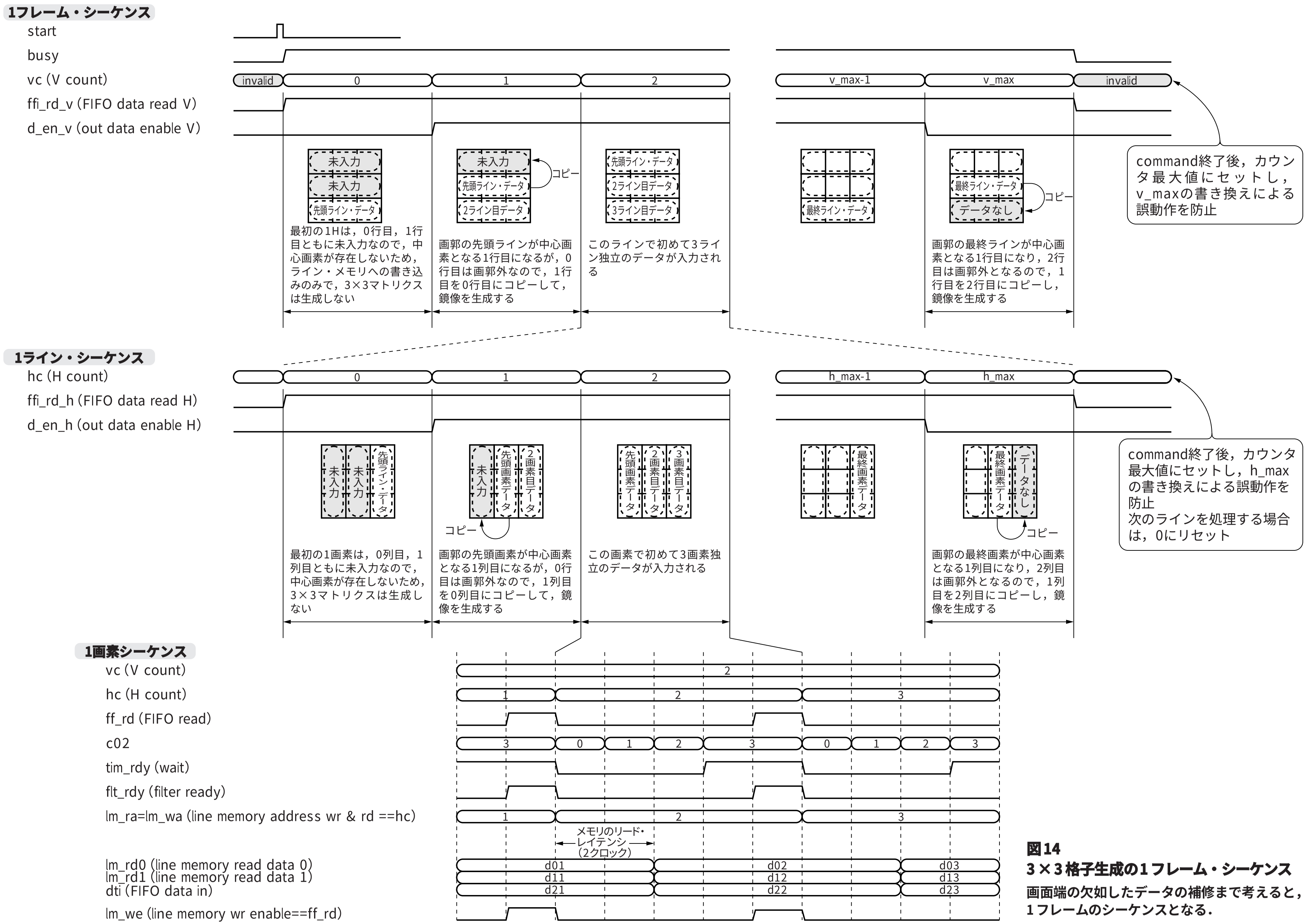


図14
3×3格子生成の1フレーム・シーケンス
画面端の欠如したデータの補修まで考えると，1フレームのシーケンスとなる．

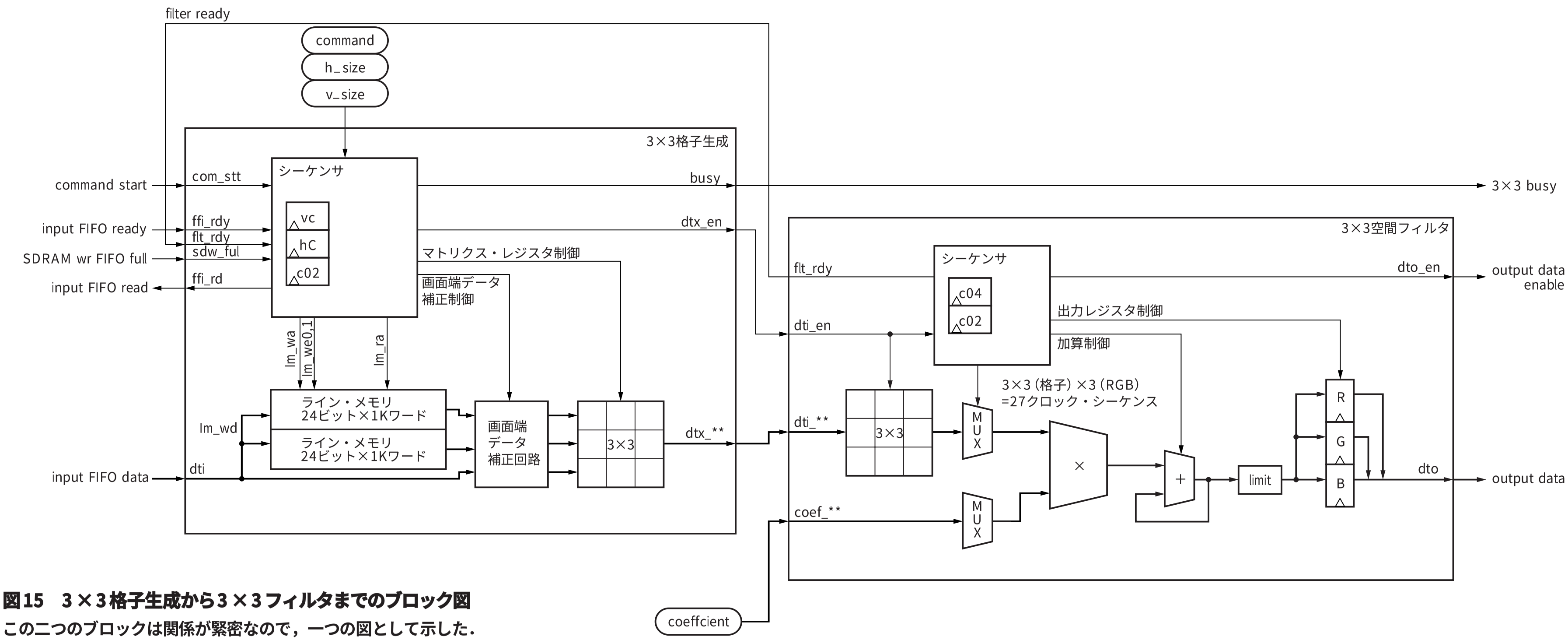


図15　3×3格子生成から3×3フィルタまでのブロック図
この二つのブロックは関係が緊密なので，一つの図として示した．

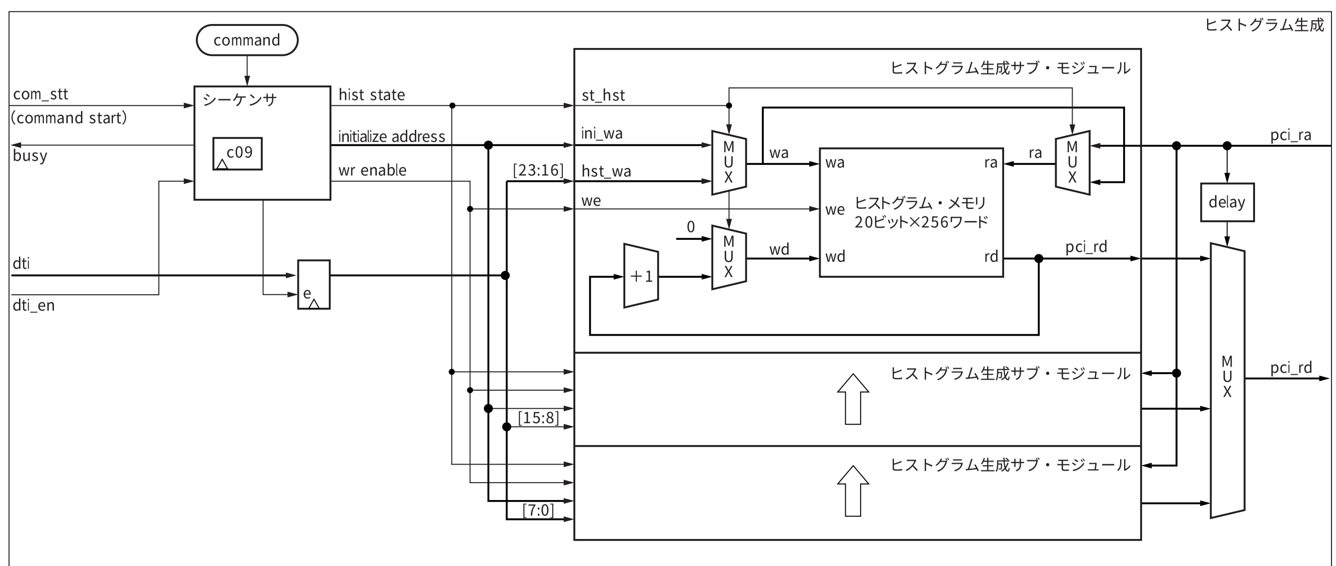
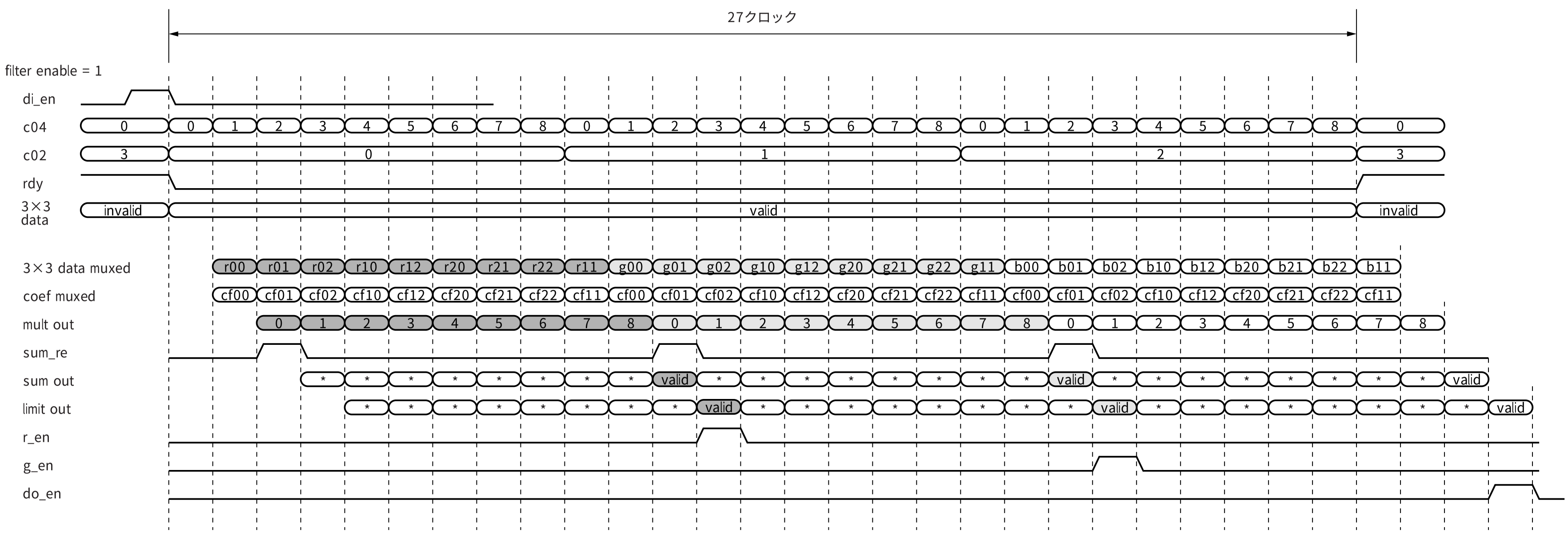


図18
ヒストグラム生成ブロック
ヒストグラム・メモリをRGB別々にする必要があり，3並列構造となる．



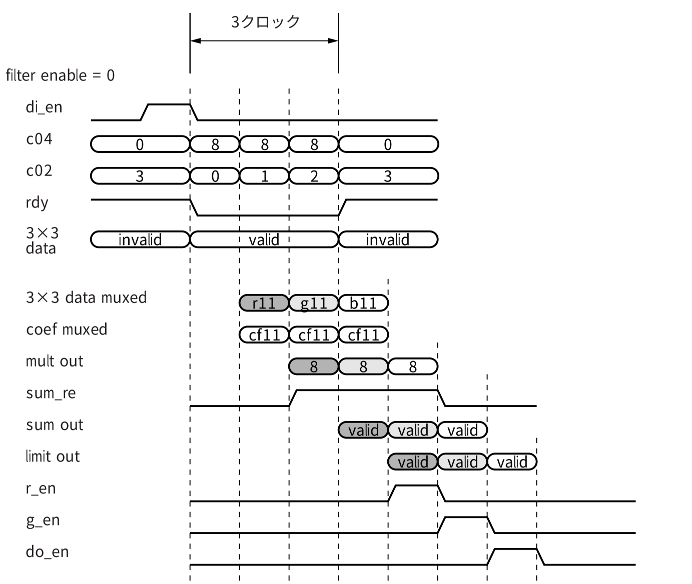


図16　3×3フィルタ演算のタイミング・チャート
フィルタONのときは27クロック・シーケンス，OFFのときは3クロック・シーケンスとなる．

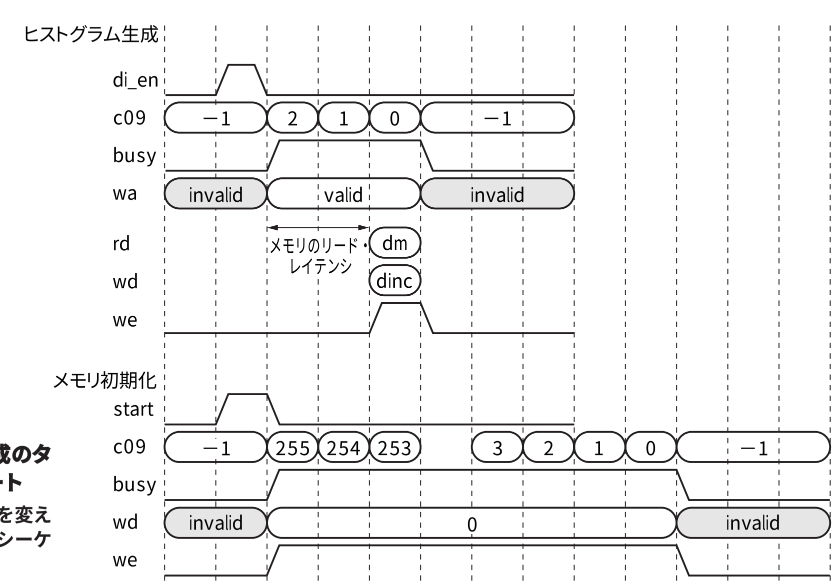


図17
ヒストグラム生成のタイミング・チャート
カウンタの初期値を変えることで，異なるシーケンスを実現する．

データ・フロー制御で述べた条件を満足させるためです．同時に，フィルタをかけないデータを通過させるための別のパスを用意する必要がなくなるので，若干ですが回路規模の節約にもなります．

図16に，フィルタのON/OFFによる27クロックと3クロックのシーケンスを，タイミング・チャートで示します．ブロック図は，**図15**の右側のブロックになります．

図15からわかるとおり，3×3格子生成と3×3フィルタの両方に3×3格子のレジスタを持っています．3×3フィルタの動作から考えると，9個の画素データは同時に必要となることはなく，順次1画素ずつ入力できればよいのです．なので，両方に3×3格子のレジスタを持つことはむだなことなのですが，今回はそれぞれのブロックが独立したIPコアとして成立するようにと考え，あえてこの方式を採用しました．例えば，3×3フィルタを並列の乗算器を持つ構造に変更したければ，3×3フィルタのみ変更するだけで格子生成はそのまま使えるわけです．とことん回路のぜい肉を落とすのであれば，二つの機能ブロックは不可分と考え，専用のシーケンスで結合されることになります．

● ヒストグラム生成 ── RGBの3系統を並列動作させる

ヒストグラム生成ブロックで必要とされる機能は以下の二つです．

- 濃度度数用メモリ(ヒストグラム・メモリ)の初期化
- 入力された濃度による度数のインクリメント

度数の初期値は‘0’なので，ヒストグラムをとる前にはかならず初期化動作が必要となります．これがヒストグラム・メモリの初期化になります．ヒストグラムの初期化コマンドが設定されスタートすると，ほかのモジュールは停止している状態なので，完全にこのモジュール単体での自走動作になります．画像データの階調が8ビット＝256階調なので，一つのヒストグラム・メモリは256のアドレスになります．したがって，256クロックのシーケンスで各アドレスに0データを書き込んでいきます．データ幅は1,024×1,024サイズの画面をカウントするために20ビットとしています．

度数のインクリメントは，本来のヒストグラムをとる操作になります．入力された濃度が指し示すアドレスのデータを読み出し，インクリメント(＋1)して，元のアドレスに書き戻します．今回のターゲット・デバイスのメモリを読み出すのに2クロック，書き戻しに1クロックを割り当てて，計3クロックのシーケンスとしています．

シーケンサはc09のカウンタで生成され，初期化の場合は255，度数のインクリメントの場合は2をカウンタにロードし，これをデクリメント(－1)して－1になればシーケンス終了となります(**図17**)．

ヒストグラム・メモリは，RGBそれぞれ独立して動作するため，3系統並列に持つ必要があります．したがって，3系統を時分割で動作させることには意味がなく，ここでは3並列同時動作となります(p.75の**図18**)．

● レベル変換 ── 処理はメモリを読み出し出力するだけ

レベル変換については，機能そのものは設定されたテーブルの内容にあるので，ハードウェアとしては入力された画素データでメモリを読み出し，これを出力するだけです．テーブル内容はRGB別々なので，やはり3並列になり，メモリを読み出すだけの2クロック・シーケンスになります．

レベル変換を行わない場合，入力＝出力というテーブルを設定することでも実現できます．しかし，レベル変換のON/OFFのたびにテーブルを設定し直すのでは，実機デバッグを行う際に不便であろうと考え，レベル変換を行わない場合は入力をメモリ読み出しパスとは別のパスで出力に抜けるようにしました．その場合には1クロック・シーケンスとなります．

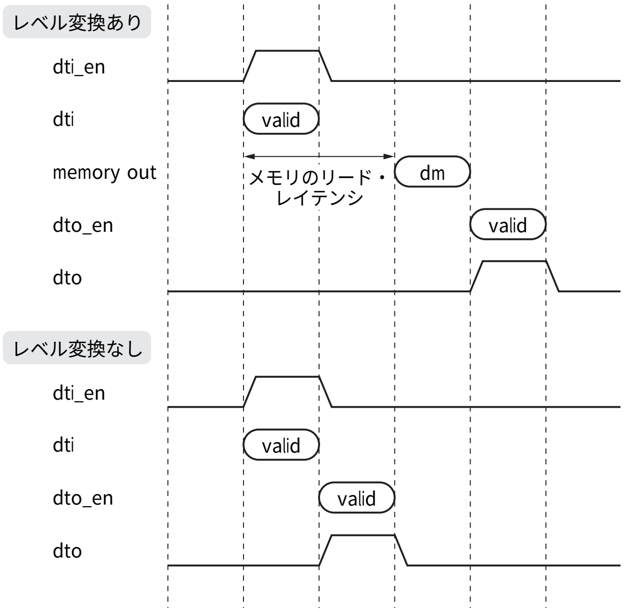


図19　レベル変換タイミング・チャート
特定のシーケンサは持たない．

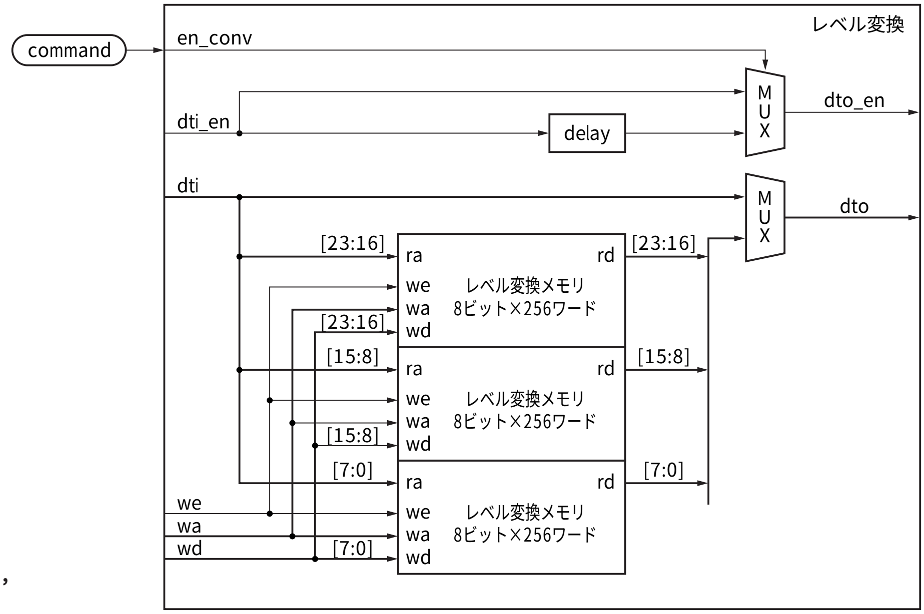


図20
レベル変換ブロック図
レベル変換テーブルをRGB別々にする必要があり，やはり3並列構造となる．

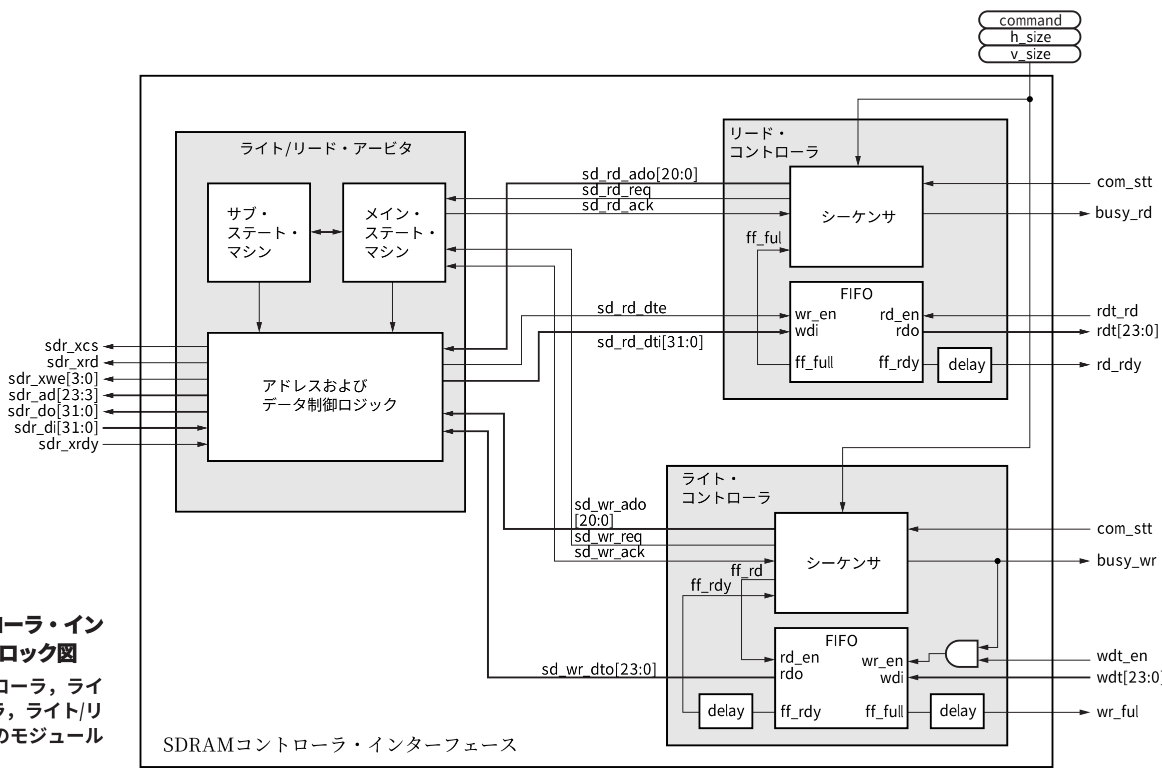


図21
SDRAMコントローラ・インターフェースのブロック図
リード側のコントローラ，ライト側のコントローラ，ライト/リード調停用の三つのモジュールからなる．

なお，シーケンスと言っていますが，単に入力のデータ・イネーブルにディレイをかけて出力しているだけなので，シーケンサらしきものは存在しません（**図19**，**図20**）．

● SDRAMコントローラI/F ──サブ・モジュールは三つ

SDRAMコントローラ・インターフェース（I/F）の基本的な要求仕様は，SDRAMコントローラとシェイクハンドしてデータを読み書きするだけです．**図21**に示すように，本モジュールは三つのサブ・モジュールからなります．各

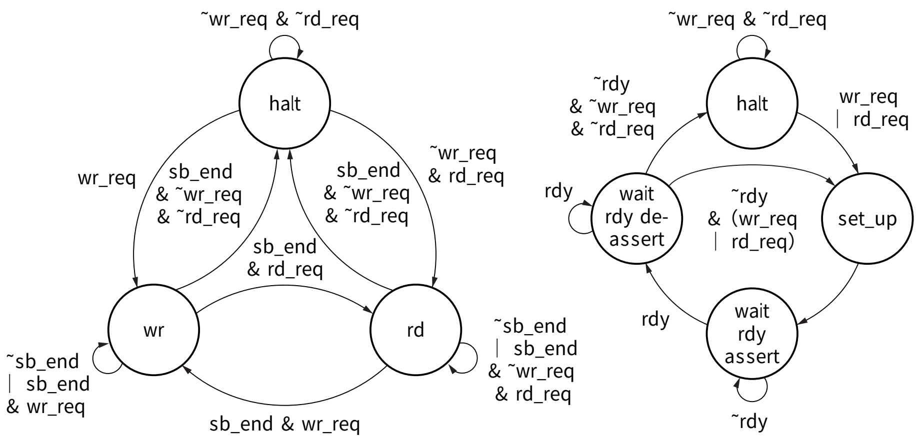


図22
SDRAMコントローラ・インターフェース状態遷移図
メインとサブの2層のステート・マシンからなる.

サブ・モジュールの機能は以下のとおりです.

● リード・コントローラ

リード・コントローラは，p.71に示した**図7**の池から水を取り出すための水門と貯水タンクにあたり，後段の要求に応じるためにデータを用意する部分です．スタート・パルスで初期化された後，v_size，h_sizeのパラメータで設定された画面のサイズ分だけライト/リード・アービタに対してデータを要求し，取り出したデータをFIFOメモリに書き込みます.

● ライト・コントローラ

ライト・コントローラは，**図7**において池に水を戻すための水門と貯水タンクにあたり，信号処理済みデータをSDRAMに書き込みます．スタート・パルスで初期化された後，信号処理系の最後段が吐き出したデータをFIFOメモリに貯め，ライト/リード・アービタに対してデータ書き込みを要求し，データを引き渡すという操作を，v_size，h_sizeのパラメータで設定された画面のサイズ分だけ繰り返します.

● ライト/リード・アービタ

リード・コントローラ，ライト・コントローラからの要求を調停し，SDRAMコントローラに対して所定のプロトコルでアクセス要求を行います．制御はメイン・ステートとサブ・ステートの二つのステート・マシンによって行われます．メイン・ステートによってライト/リードの調停を，サブ・ステートによってSDRAMコントローラに対するアクセスを行います．それぞれのステート・マシンの状態遷移図を**図22**に，SDRAMコントローラに対するアクセスのタイミング・チャートを**図23**に示します.

● PCIバス経由でのデータの書き込みと読み出し

このモジュールの基本仕様は，次の三つです.

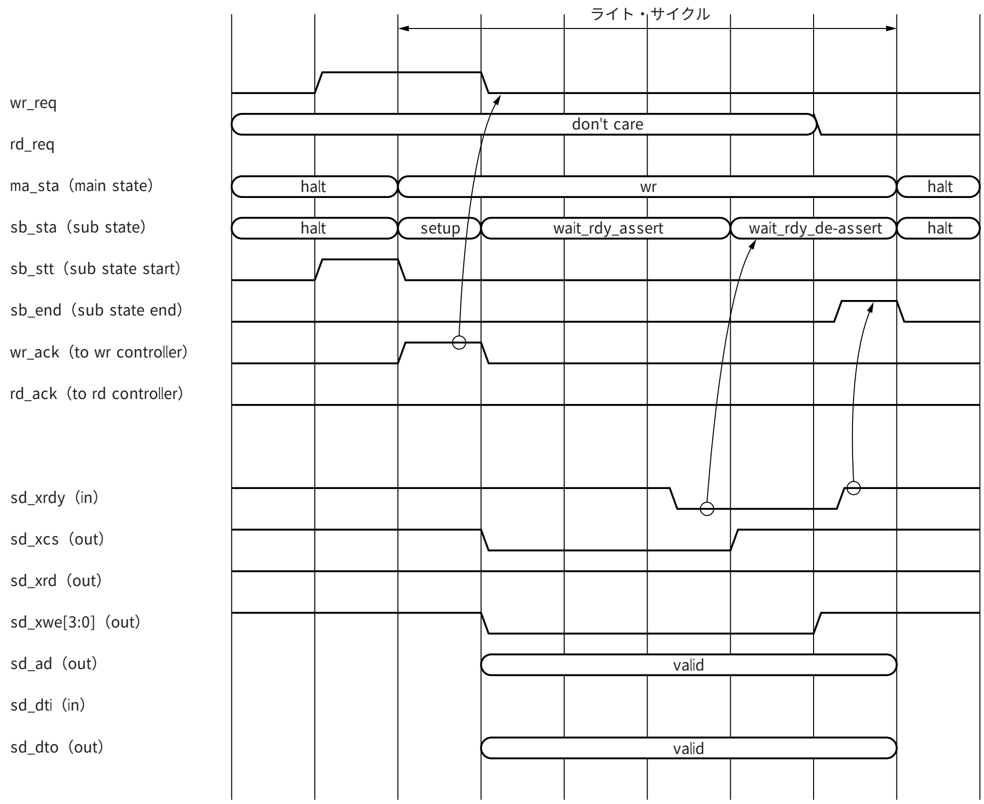
- パラメータ・レジスタの設定(ライト)
- レベル変換テーブルの設定(ライト)
- ヒストグラムの結果の読み出し(リード)

パラメータ・レジスタ，レベル変換テーブルの設定については，PCIバスからのアドレスをデコードし，指定されたアドレスにデータを書き込むだけです．ヒストグラムの結果の読み出しについても，指定されたアドレスのデータがヒストグラム生成モジュールから出てくるので，これを通過させるだけの機能となります(**図24**).

4 トップ・レベル・シミュレーションのためのヒント

各モジュールにおける機能検証が適正に行われているならば，画像処理IPコアとしてのトップ・レベル検証では個々のモジュールの機能を詳細に検証する必要はありません．例えば，フィルタが正しくフィルタとして機能するかどうかは，フィルタ・モジュールの機能検証で行うべき作業であり，トップ・レベルの検証では「フィルタは単体では正しく機能する」という状態になっている必要があります.

また余談になりますが，筆者がフィルタを検証する場合は，フィルタとして正しく機能しているかという視点では検証を行いません．単純に数値演算回路と割り切ってしまい，正しく演算すれば正しい答えが出るという前提で，乱数などを入力し，C言語あるいはビヘイビア・モデルによる演算結果との比較で検証してしまいます．フィルタ実現



（**a**）halt→write→halt

図23
SDRAM コントローラのタイミングチャート
リクエストに応じて，ライト/リードそれぞれの動作が切り替わる．

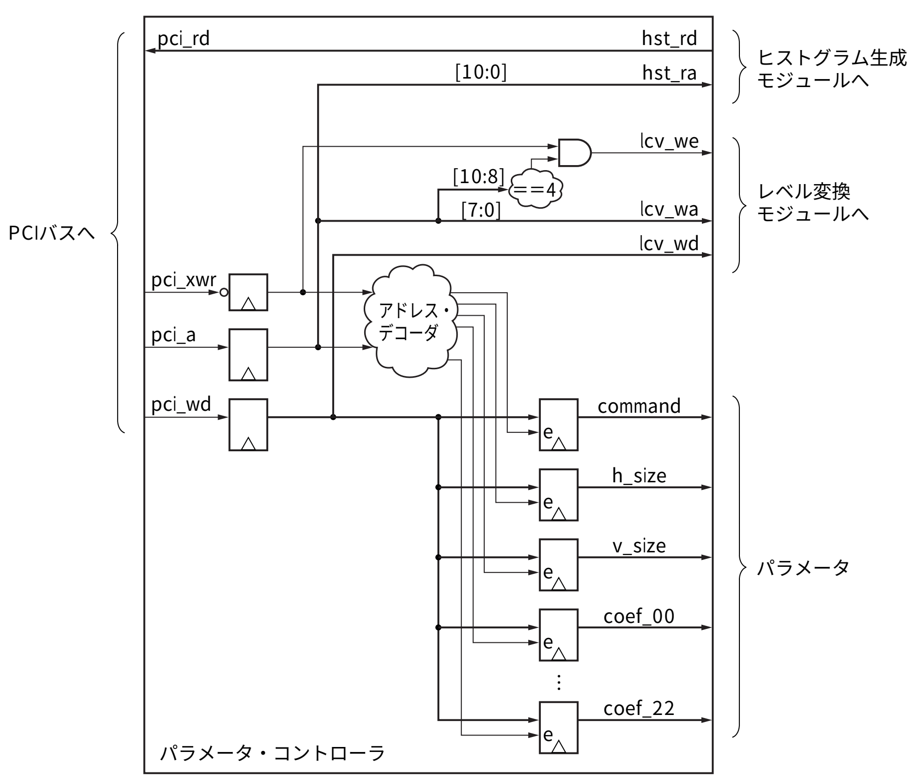
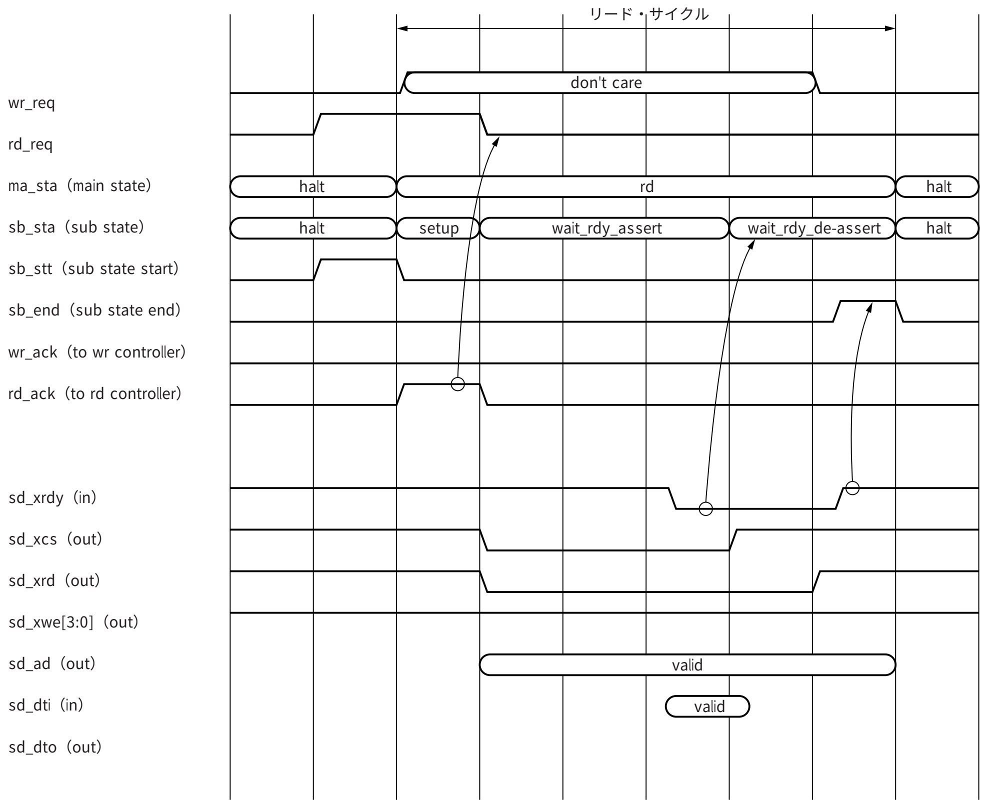


図24
PCI バス経由でのデータ・ライト/リードのブロック図
パラメータの書き込みとヒストグラム読み出しのみの単純な構造をとる．



（**b**）halt→read→halt

リスト1
メモリに初期値を与える

```
define          PATRN     "C:/personal/picture/lena.rgb"

module mt48lc2m32b2_inipic_b0 (Dq, Addr, Ba, Clk, Cke, Cs_n, Ras_n, Cas_n, We_n, Dqm);
......
// modified by s.suzuki
    integer         c_x    ,c_y ;
    reg      [23:00] dt_tmp ;
    reg      [23:00] buffi  [0:V_size*H_size-1];
    initial begin
        $readmemh (`PATRN,buffi );
        for ( c_y=0; c_y<V_size; c_y=c_y+1 ) begin
            for ( c_x=0; c_x<H_size; c_x=c_x+1 ) begin
                // インクリメント・データを使用する場合
                Bank0[c_y*1024+c_x] = c_y*1024+c_x ;
                // 画像データを使用する場合
                dt_tmp = buffi[c_y*H_size+c_x] ;
                dt_tmp = {dt_tmp[07:00],dt_tmp[15:08],dt_tmp[23:16]} ;
                if     ( c_y<512 ) Bank0[c_y*1024+c_x]       = dt_tmp ;
                else               Bank1[(c_y-512)*1024+c_x] = dt_tmp ;
            end
        end
    end
```

における数値演算が正しいかどうかはアルゴリズムの問題であり，RTL検証で行う作業ではないと考えています．

話がそれてしまいましたが，そういうわけでトップ・レベル検証において行う検証項目は，

- 各モジュールのI/Oの結合
- 全体としての制御
- 適正なデータの通過

など，意外に少ないものです．それでも，複数人でRTLを持ち寄って検証する場合などは，なかなか緊張感があるものです．

● ベンダ提供のシミュレーション・モデルにひとくふう

今回の仕様では，画像データがすでにSDRAM上に書き込まれていることになっているので，その状況を作り出し

リスト2　FPGA内蔵RAMのシミュレーション・モデル

```
// histgram memory
`ifdef SIMULATION
    piccopro_mema  #(
                    8   ,       // address width
                    256 ,       // address depth
                    20  )       // data width
             um     (
             .rd    (mem_rd ), // read data out

             .wd    (mem_wd ), // write data in
             .wa    (mem_wa ), // write address
             .we    (mem_we ), // write enable
             .wclk  (clk    ), // write clock
             .ra    (mem_ra ), // read address
             .rclk  (clk   )); // read clock
`else
    wire     [31:00] mem_wd_tmp = { 12'h000,mem_wd } ;
    wire     [31:00] mem_rd_tmp;
    assign           mem_rd = mem_rd_tmp[19:00] ;
    piccopro_m4k_32x256 um (
                 .data       (mem_wd_tmp ),
                 .wren       (mem_we     ),
                 .wraddress  (mem_wa     ),
                 .rdaddress  (mem_ra     ),
                 .clock      (clk        ),
                 .q          (mem_rd_tmp ));
`endif
```

てやらなければなりません．といって，ホストからのデータの書き込みをシミュレーションするのも大げさなので，SDRAMのシミュレーション・モデル（米国Micron Technology社提供）を改造して，シミュレーション初期化の段階でメモリにデータを入れてしまいました．**リスト1**に示すとおり`initial`文を追加して，`reg`宣言されているメモリに初期値を与えています．「インクリメント・データを使用する場合」と「画像データを使用する場合」のいずれかを適用することになります．インクリメント・データは，検証の初期段階で正しくデータがデータパスを通過しているかを検証するのに便利です．実際の画像データを使用する場合，1フレームのデータが適正に処理されているかを全体像として評価するのに適しています．

半導体メーカが提供しているシミュレーション・モデルに手を加えるのに抵抗がある方も少なくないでしょうが，全部読んで理解する必要はありません．今回の場合であれば，メモリ・セルを宣言している`reg`文を探し出し，初期化されていないことが確認できればしめたものです．個人で使用する分には問題ないと思いますので，必要に応じて挑戦してみてはいかがでしょうか．

今回のRTLにおいてターゲット・デバイスのエンベデッドRAMブロックを使用している箇所では，**リスト2**のような`` `ifdef SIMULATION``の記述が各所にあります．シミュレーション時にSIMULATIONというテキスト・マクロを定義することで，`` `ifdef``側に記述してある機能が有効になります．`` `ifdef``側にはビヘイビア記述のRAMモデルが，`` `else``側にはQuartus II（Altera社のFPGA開発ツール）のMega Function Wizardによって生成されるモデルが記述されています．

Altera社から提供されているRAMモデルは，いろいろな意味でまじめに作られているので，シミュレーションを行う際にはこのモデルが大きな負荷になります．こうした場合，ビヘイビア記述のモデルを代用することで負荷を軽減し，シミュレーション速度を上げることができます．もちろん，提供されているRAMモデルとの等価性をあらかじめ評価しておく必要はありますが，たくさんのRAMを使用する場合には効果絶大ですので，試してみてください．

● 作業を効率良く進めるためのダミー・モデルの作成

トップ・レベル検証とは関係ないのですが，今回のシステムを作成する際に共同作成者（第3章を執筆した森岡氏）から「画像処理IPコアのダミー・モデル（piccopro_top_dummy.v）を提供してもらえないか」との要求を受けました．この時点で筆者側のRTLがしあがっていれば，それを渡せばよいだけなのですが，作業を開始したばかりでそうもいきません．同氏の要求は，

- PCIバス経由のホストからのライト/リードの確認
- スタート・パルスによるコマンド実行とレディ制御の確認

ということだったので，これを実現できるようなモデルをでっち上げて渡すこととしました．でっち上げると言っても，ソフトウェアを含めての実機検証を行うことになるため，ここでは論理合成可能なモデルであるということが重要になります．

システムを作り上げていく過程では，このような最終的には使用しないモデルの作成などが必要になることもあります．今回は，これによってお互いの手を止めることなく並行して作業を進められたと同時に，SDRAMを除いて外部とのやり取りがあらかじめ実機上で確認できたことになります．このようにモデル作成のメリットは大きく，期間の長いプロジェクトではかならずと言っていいほど必要になるので，めんどうがらずに手を動かしましょう．筆者も，つい「え～」と声が出てしまうほうなのですが…（森岡さん，ごめんなさい！）．

Column ヒストグラムの平たん化の方法

画像処理IPコア内には，ヒストグラム命令の実行によりヒストグラムが自動的に生成されています．RGBで各256階調なので，256アドレスのメモリ3個にそれぞれ度数が保存されています．ヒストグラムをとる画像領域は最大1,024×1,024としたので，度数の最大値は1,048,576となり21ビットのデータになりますが，さすがに全面が単一色であることはないと考え，20ビットとしています．まずは，20ビット×256×3のヒストグラム・データをFPGA内のメモリから読み出します．次に，ヒストグラムの累積度数をとります．各色に対して256階調それぞれの累積度数を求めます．n階調目の度数を$d(n)$，累積度数を$sum(n)$とすると，以下の式で求められます．

$$sum(n) = \sum_{k=0}^{n} d(k)$$

続いて，ヒストグラムを平たん化するためのテーブル値を求めます．各色に対して256階調それぞれのテーブル値を求めます．n階調目のテーブル値を$tbl(n)$とすると，以下の式で求められます．

$$tdl(n) = \mathrm{int}\left(\frac{sum(n) - sum(0)}{sum(255) - sum(0)} \times 255\right)$$

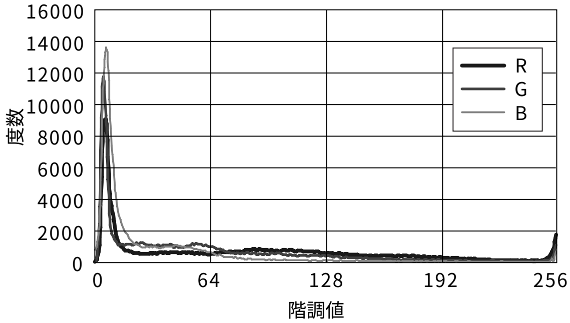
ちなみに，

$sum(0) = d(0)$
$sum(255)$ ＝画面の画素の総数

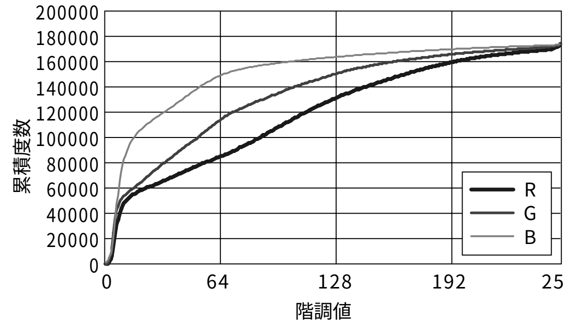
となります．

最後に，こうして求められたテーブルを，IPコア内のテーブル・メモリに書き戻し，テーブル変換命令を実行すると，ヒストグラムは平たん化され，コントラスト補正された画像ができ上がります．

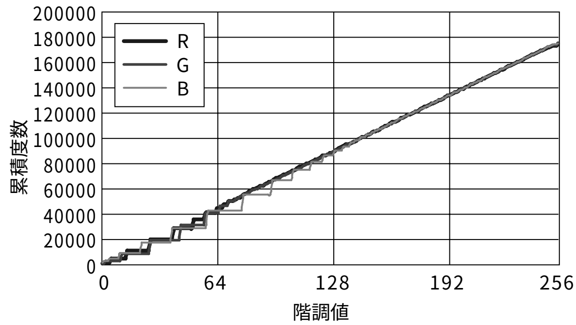
図A-1に本文の**写真1(a)**を使ってヒストグラムを作成した結果を示します．これをもとに累積度数をとったものが**図A-2**になります．ヒストグラムの平たん化を行った後に再度ヒストグラムを作成し，累積度数をとり直したものを**図A-3**に示します．ヒストグラムが平たん化されたことで，累積度数が直線になっているのが確認できます．なお，平たん化された後のヒストグラムは階調が抜けている部分があるため，度数が0となる部分が断続的に発生します．そのため，非常に見づらいものとなってしまうので，累積度数で表現することとしました．



図A-1　サンプル画像のヒストグラム



図A-2　サンプル画像の累積度数



図A-3　サンプル画像のヒストグラム平たん化後の累積度数

参考・引用*文献

(1) 谷口慶治；画像処理工学基礎編，共立出版，1996年11月．
(2) 貴家仁志；連載 よくわかる動画・静止画の処理技術，Interface，1998年2月号～11月号．
(3) 酒井幸市；ディジタル画像処理の基礎と応用，CQ出版社，2003年9月．

すずき・しょうじ

<筆者プロフィール>
鈴木昌治．地方（新潟県）在住のため，（今回設計した画像処理システムについての）森岡氏とのやりとりは最初のざっくりとした打ち合わせ以外はすべてメールによるものでした．問題なく動作して正直ほっとしてます．